

Gestione del Processore *(Scheduling)*

Scheduling dei processi

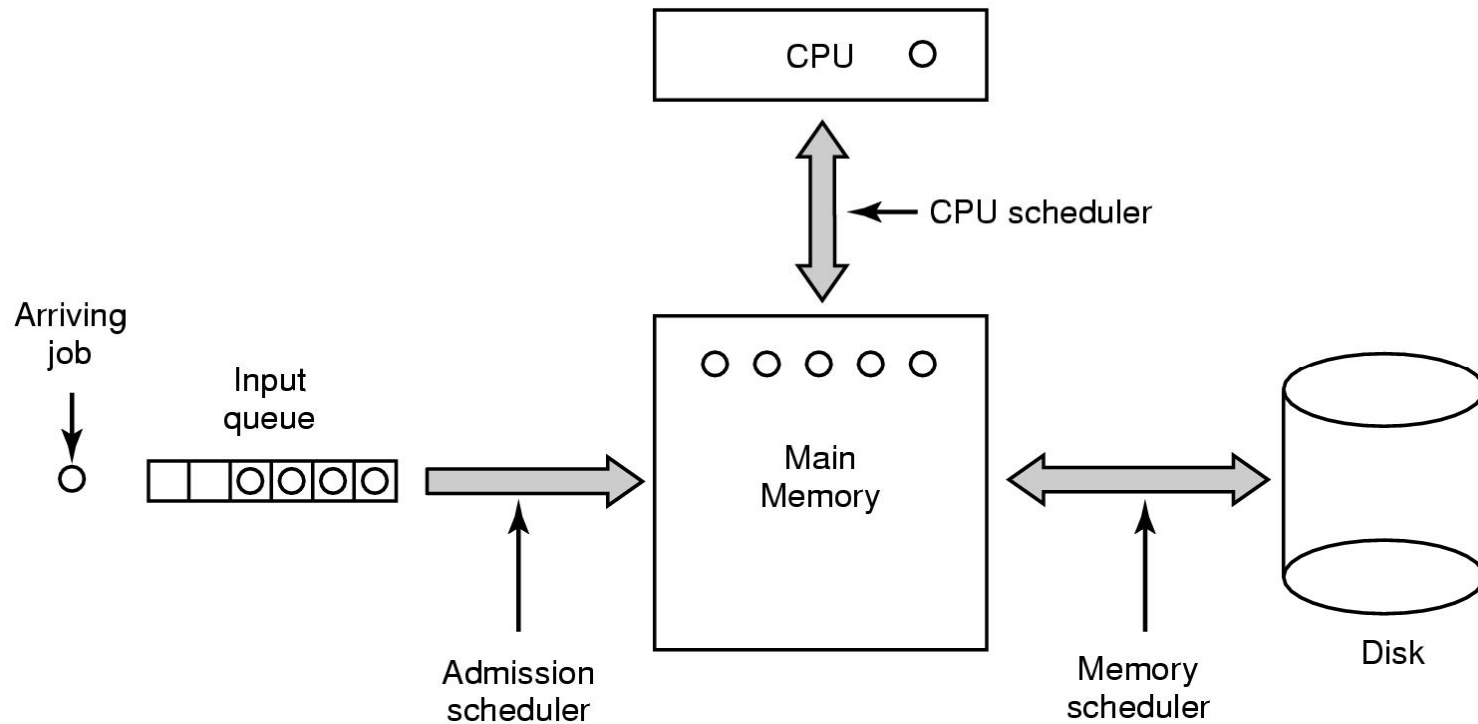
- È l'attività mediante la quale il sistema operativo effettua delle scelte tra i processi, riguardo:
 - all'assegnazione del processore
 - al caricamento in memoria centrale

Tre diversi livelli di scheduling:

- **Scheduling a breve termine**
 - Sceglie tra i processi pronti quello a cui assegnare il processore
 - Interviene quando un processo esce dallo stato di esecuzione
 - non preemptive scheduling
 - preemptive scheduling
- **Scheduling a medio termine (Swapping)**
 - trasferimento temporaneo di processi in memoria secondaria
 - disponibilità di memoria principale inferiore alla richiesta complessiva dei processi
- **Scheduling a lungo termine**
 - Sceglie i nuovi job da caricare in memoria
 - Controlla il grado di multiprogrammazione
 - È una componente importante dei sistemi batch multiprogrammati

Scheduling nei sistemi Batch

Tre livelli di scheduling



Short-term Scheduling (1)

- Lo *scheduler* si occupa di decidere quale fra i processi *pronti* può essere mandato *in esecuzione*
- L'algoritmo di scheduling ha impatto su:
 - efficienza nell'utilizzo delle risorse della macchina
 - prestazioni percepite dagli utenti
- Lo scheduling ha obiettivi diversi in diversi sistemi (batch, interattivi...)

Obiettivi dello Scheduling

Obiettivi principali degli algoritmi di scheduling:

- *Fairness* (Equità) - processi dello stesso tipo devono avere trattamenti simili
==> In particolare: evitare l'*attesa indefinita*
- *Balance* (Bilanciamento) - tutte le parti del sistema (CPU, dispositivi ...) devono essere utilizzate al massimo (sistemi batch)
- *Throughput* - massimizzare il numero di job completati in un intervallo di tempo (sistemi batch)
- *Turnaround time* - minimizzare il tempo di permanenza di un job nel sistema (sistemi batch)
- *Tempo di risposta* - minimizzare il tempo di risposta agli eventi (sistemi interattivi)
- *Proporzionalità* - assicurare un tempo di risposta proporzionale alla complessità dell'azione (sistemi interattivi)

Short-term Scheduling (2)

- Scheduling senza prerilascio (*non preemptive*)
 - lo scheduler interviene solo quando un processo viene creato, oppure termina, oppure si blocca per effetto di una SVC o di un evento esterno
- Scheduling con prerilascio (*preemptive*)
 - lo scheduler può intervenire ogni volta che è necessario per ottenere gli obiettivi perseguiti
 - quando diventa *pronto* un processo a più alta priorità rispetto a quello *in esecuzione*
 - quando il processo *in esecuzione* ha sfruttato la CPU per massimo tempo consentito (*quanto di tempo*)

Short-term Scheduling (3)

- Due tipologie di processi :
 - processi *CPU-bound* -- lunghi periodi di elaborazione fra due richieste successive di I/O
 - processi *I/O-bound* -- brevi periodi di elaborazione fra due richieste successive di I/O
- Se l'obiettivo è il *Turnaround Time*: dare priorità ai processi *I/O-bound*

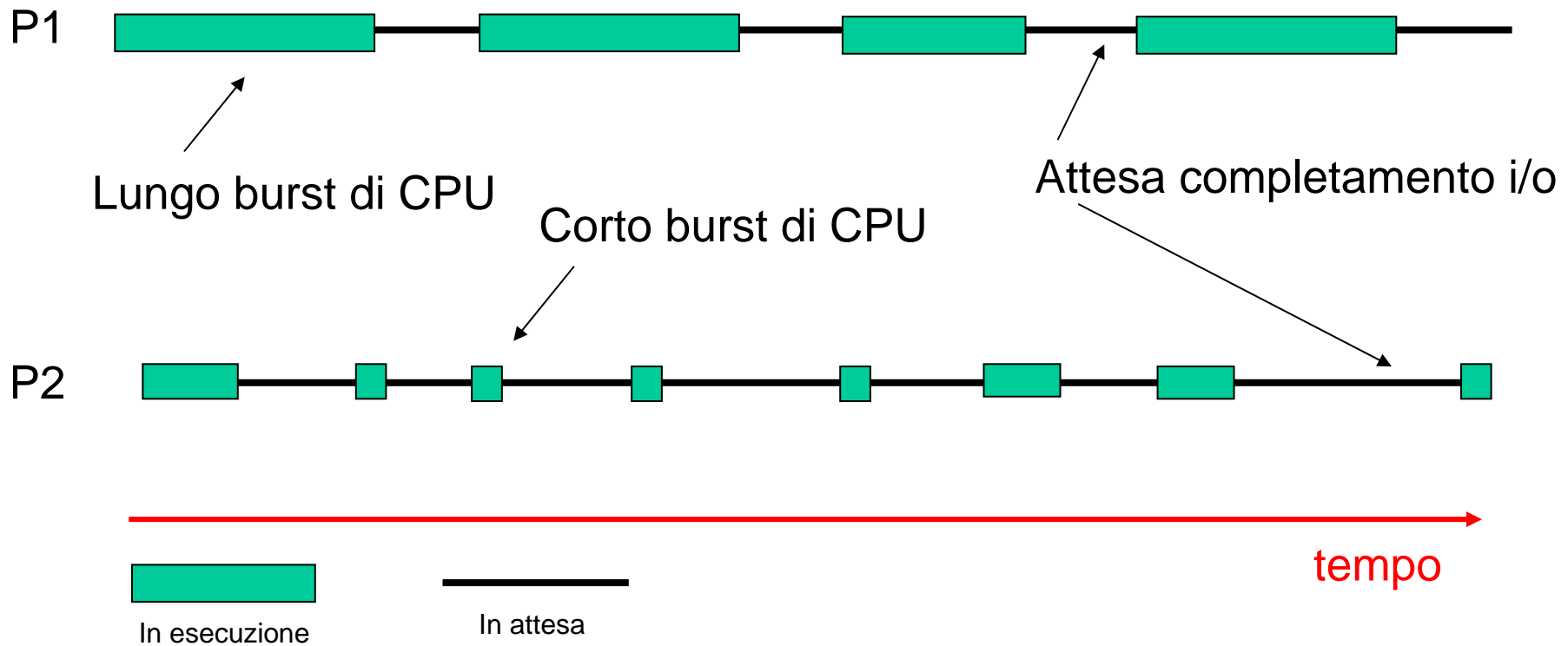
Politiche di Scheduling

- FIFO
- Priorità
- SJF (shortest job first)
 - Sistemi batch
- SRTF (shortest remaining time first)
- Round Robin
 - Sistemi *interattivi*
- Code Multiple
 - Sistemi batch/interattivi

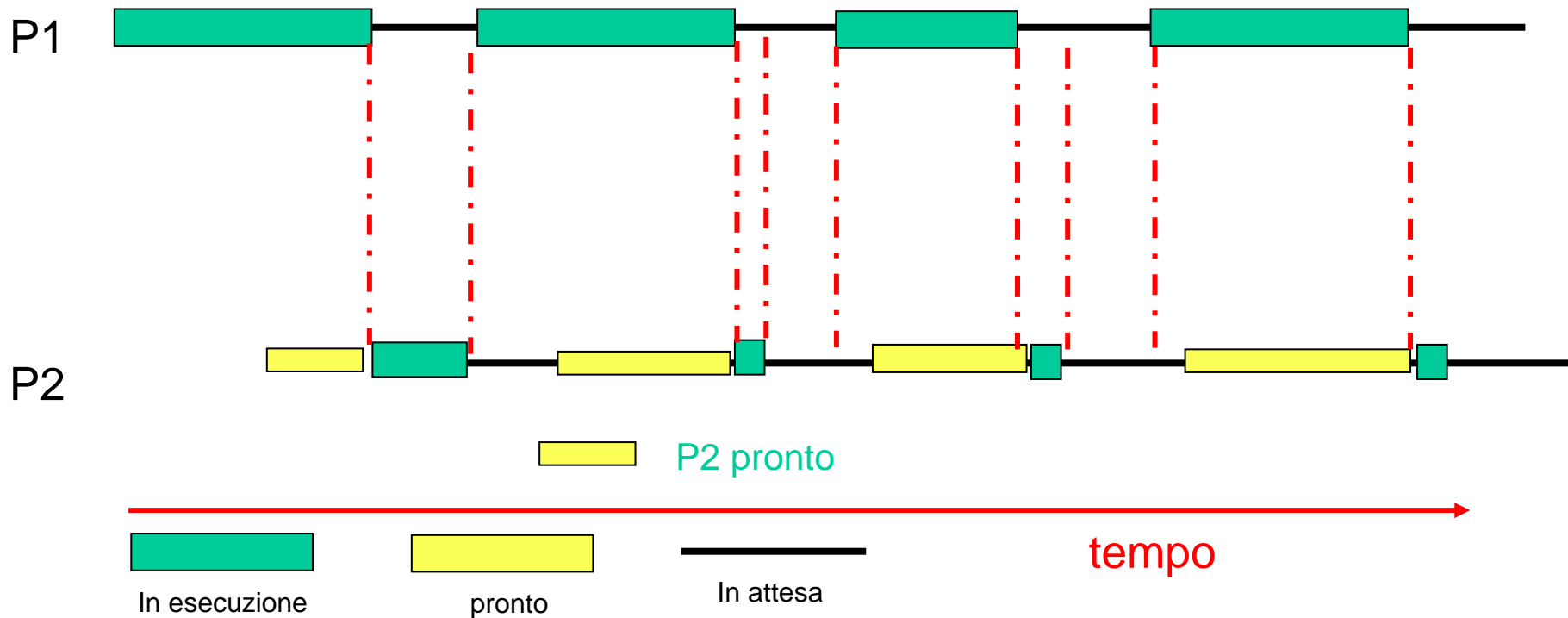
Scheduling con politica FIFO

- I processi non hanno priorità
- La coda dei processi pronti è una coda FIFO
- La politica non prevede prerilascio
==> lo scheduler interviene solo quando il processo in esecuzione si blocca per effetto di una SVC o di un evento esterno, oppure termina
- Ad ogni intervento, lo scheduler assegna il processore al primo dei processi pronti
- Non esiste possibilità di attesa indefinita
- Se sono presenti processi CPU-bound e processi I/O-bound, la politica FIFO penalizza i processi I/O-bound (aumentando il tempo di turnaround)

Processi *CPU bound* (P1) e *I/O bound* (P2)

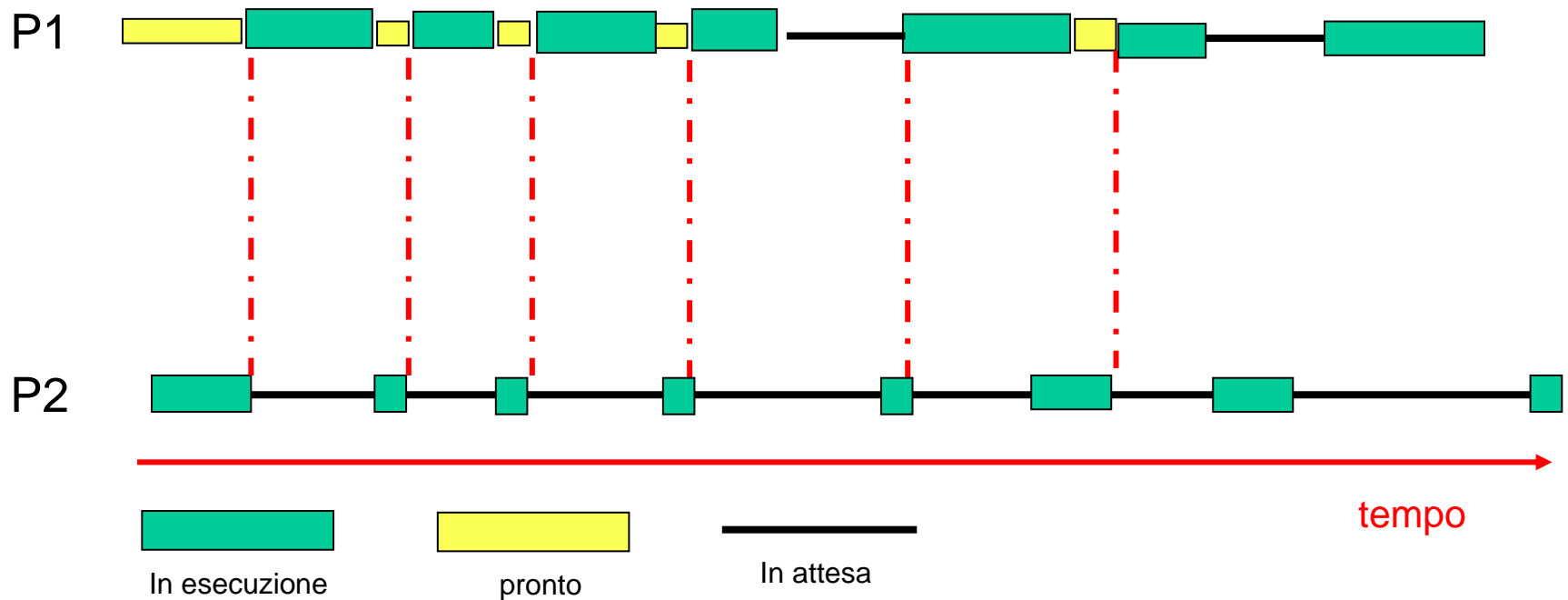


Processi *CPU bound* (P1) e *I/O bound* (P2)



1) FIFO, senza prerilascio

Processi *CPU bound* (P1) e *I/O bound* (P2)



2) Priorità a *I/O bound*

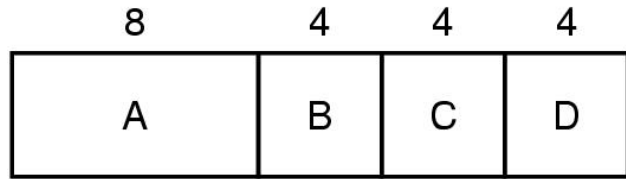
Scheduling con priorità (1)

- Ogni processo ha una priorità
- Ogni volta va in esecuzione il processo a priorità più elevata
- Punti chiave :
 - Priorità statica
 - ==> Possibilità di **attesa indefinita** per i processi a priorità più bassa
 - Priorità dinamica
 - ==> strategia di assegnazione della priorità ?
 - per esempio: priorità maggiore ai processi I/O bound, ma come individuare i processi I/O bound?*

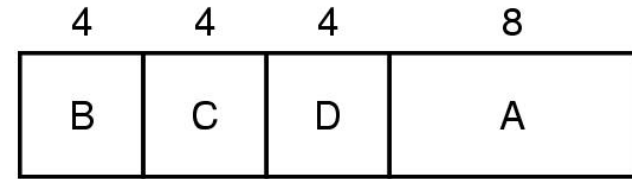
Scheduling con priorità (2)

- Molte strategie per il calcolo dinamico della priorità:
 - Priorità crescente nel tempo per i processi che rimangono in stato di pronto. Obiettivo: impedire *attesa indefinita*
 - politica *Shortest Job First (SJF)*. Obiettivo: *turnaround*
 - Incremento di priorità quando i processi vengono *riattivati*. Obiettivo: *tempo di risposta*
 - Priorità legata alla percentuale f del quanto di tempo che è stato consumato l'ultima volta che il processo è andato in esecuzione (es. proporzionale a $1/f$). Obiettivo: favorire processi *I/O bound*

Politica SJF (1)



(a)



(b)

Scheduling con politica *Shortest Job First (SJF)*

==> privilegia il job più corto

Ipotesi:

- l'insieme dei job da schedulare è noto all'inizio
- si conosce il tempo di esecuzione T di ogni job

Politica:

- i job sono schedulati in ordine di T crescente
- non c'è prerilascio

Proprietà: SJF minimizza il tempo medio di turnaround

Politica SJF (2)

SJF minimizza il tempo *medio* di *turnaround*

Infatti:

4 job A,B,C,D con tempi di esecuzione a, b, c, d

Sequenza di schedulazione: $a \rightarrow b \rightarrow c \rightarrow d$

- turnaround(A) -- a
- turnaround(B) -- $a + b$
- turnaround(C) -- $a + b + c$
- turnaround(D) -- $a + b + c + d$

turnaround totale $4a + 3b + 2c + d$

minimo quando a, b, c, d sono in ordine crescente

Politica SJF (3)

Stima della durata del *CPU Burst* con *media esponenziale*

- Ogni processo avanza eseguendo una sequenza di *CPU burst* (intervalli di esecuzione).
- Per applicare la politica SJF al *burst* corrente, si stima la durata (incognita) $s(n+1)$ del burst $B(n+1)$ in base alle durate $d(j)$ (note) dei burst $B(j)$, $j \leq n$

Fissato un parametro a con $0 \leq a \leq 1$, si definisce:

$$s(n+1) = a d(n) + (1-a) s(n)$$

Espandendo:

$$s(n+1) = a d(n) + (1-a) a d(n-1) + (1-a)^2 a d(n-2) + \dots + (1-a)^{n+1} a d(0)$$

\Rightarrow Per esempio: con $a = 0,5$ la durata dei burst pregressi influiscono con peso che si dimezza a ogni decremento dell'indice.

Politica SRTF

Shortest Remaining Time First (SRTF)

Politica con priorità dinamica e revoca del processore

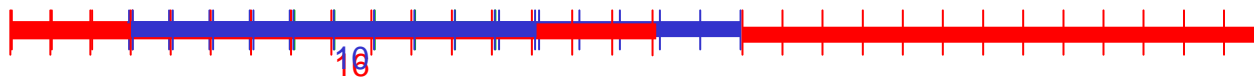
Ipotesi:

- *l'insieme dei job da schedulare è variabile nel tempo*
- *si conosce il tempo di esecuzione T di ogni job*

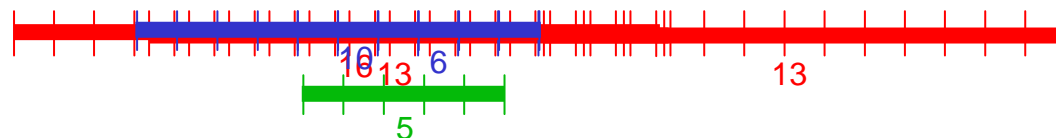
Per ogni processo pronto o in esecuzione, priorità uguale all'opposto del tempo mancante al completamento del *CPU burst*

==> nota: ha precedenza il processo con il massimo valore di priorità

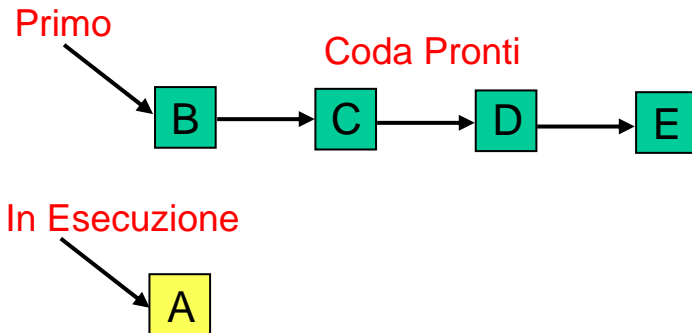
In esecuzione



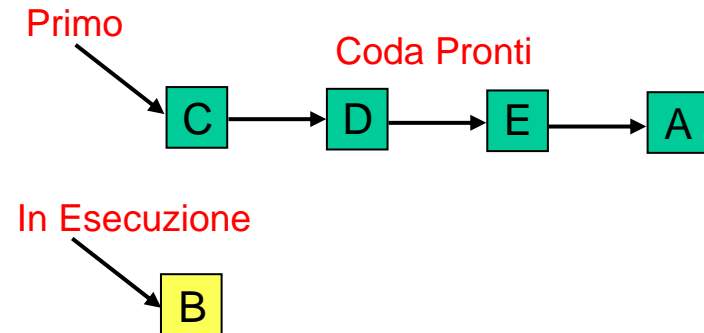
Pronti



Politica *Round Robin* (1)



(a) A passa in esecuzione



(b) A esaurisce il *quanto* di tempo

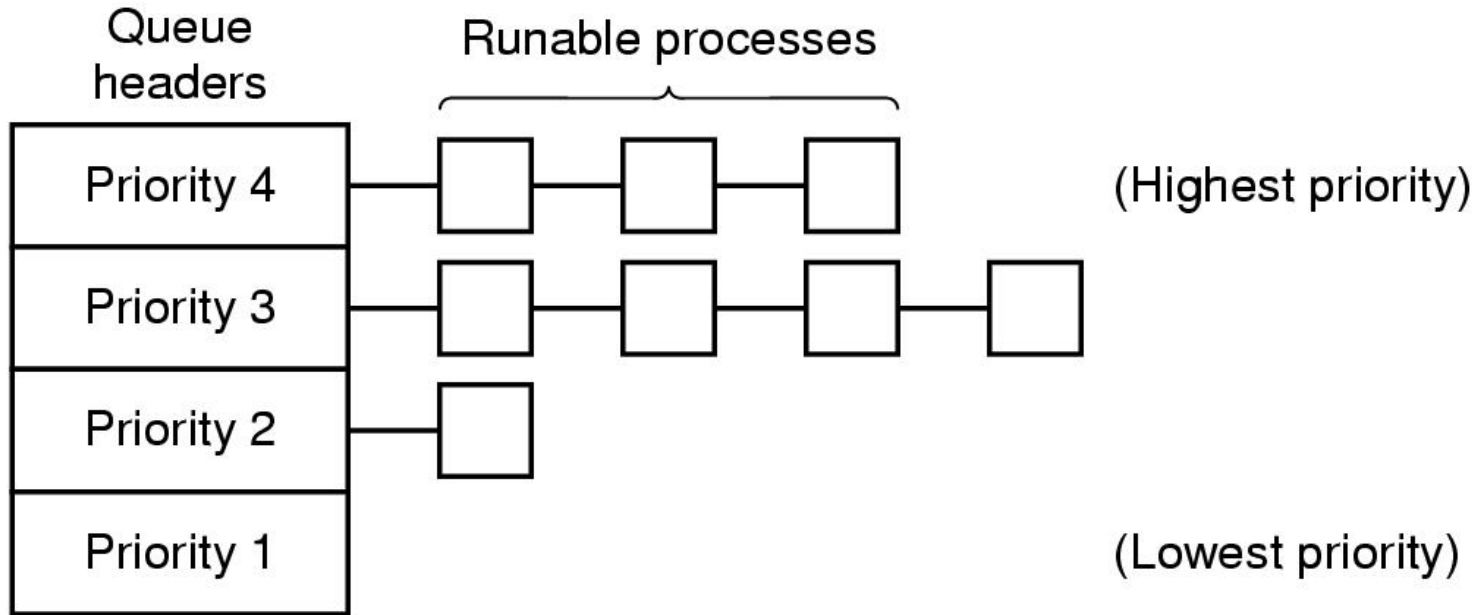
Proprietà:

- Tempo di risposta limitato superiormente
 $= (\text{Numero Proc Pronti}) * (\text{Quanto di Tempo})$
- Proporzionalità
tempo di Turnaround proporzionale a lunghezza del lavoro

Politica *Round Robin* (2)

- Esaurimento del quanto di tempo segnalato dal timer del processore
 - L'interruzione del timer provoca l'attivazione dello scheduler
 - Al termine del suo intervento, lo scheduler carica il timer con un nuovo quanto di tempo
- Lo scheduler interviene anche quando il processo in esecuzione si sospende prima della scadenza del quanto di tempo
 - lo scheduler riassegna il processore e carica il timer con un nuovo quanto di tempo
- Come fissare il quanto di tempo
 - deve essere abbastanza lungo da ammortizzare il costo di un *context switch* deve
 - essere abbastanza breve da permettere una risposta veloce agli utenti interattivi
 - in sistemi reali tipicamente 20-120 ms
- RR *non* favorisce i processi I/O bound

Scheduling con Code multiple (1)



Esempio con 4 classi di priorità

- All'interno della Classe Attiva si applica la politica Round Robin
- Prendiamo tra le classi e all'interno della Classe Attiva (Round Robin) va in esecuzione il primo processo della classe non vuota di massima priorità (*classe attiva*)

Scheduling con Code multiple (2)

- Va in esecuzione il primo processo della classe non vuota di massima priorità (*Classe Attiva*)
 - All'interno della Classe Attiva si applica la politica *Round Robin*
 - Prerilascio tra classi e all'interno della *Classe Attiva* (*Round Robin*)
- ==> Possibilità di attesa indefinita*

Eventualmente:

- Priorità dinamiche

==> si evita l'attesa indefinita

Esempio: Riduzione della priorità per i processi che esauriscono il quanto di tempo più di k volte, con k opportuno

- Valori diversificati del quanto di tempo per le diverse classi di priorità

==> si riduce l'overhead dovuto al cambio di contesto

Esempio: quanto di tempo maggiore per le classi a più bassa priorità:

Scheduling dei Thread (1)

Lo scheduling dei thread

- utilizza algoritmi simili a quelli visti finora
- viene implementato in modo diverso a seconda che i thread siano realizzati a livello kernel o a livello utente

Scheduling dei Thread (2)

- Lo scheduling dei thread *kernel level*
 - L'unità di schedulazione del kernel è il thread (non il processo)
 - quando un thread di un dato processo esce dallo stato di esecuzione, lo scheduler non è vincolato a mandare in esecuzione un thread dello stesso processo
 - *Nella seconda ipotesi il cambio di contesto ha costo maggiore, perchè cambia lo spazio di indirizzamento*
 - I quanti di tempo (se previsti dalla politica di scheduling) sono assegnati ai thread
 - Scheduler attivato dall'interruzione del *Timer*
 - Applicabili politiche a priorità con prerilascio

Scheduling dei Thread (3)

- Lo scheduling dei thread *user level*
 - L'unità di schedulazione del kernel è il processo (non il thread)
==> *il kernel ignora l'esistenza dei thread*
 - Per ogni processo, si utilizza uno *scheduler dei thread* (funzione della *thread library*, che opera in stato utente)
 - quando un thread di un dato processo esce dallo stato di esecuzione, lo scheduler dei thread è vincolato a mandare in esecuzione un thread dello stesso processo
==> *il cambio di contesto ha un costo (overhead) moderato*
 - Normalmente: lo schedulatore dei thread (anche) attivato esplicitamente dal thread che rilascia il processore (*thread_yield*)
 - Applicabili politiche a quanti di tempo
==> *difficoltà: chi riceve l'interruzione del timer?*
 - Non applicabili politiche a priorità con prerilascio